UCRL-CONF-202451

# REVERSIBLE $N$–BIT TO $N$–BIT INTEGER HAAR–LIKE TRANSFORMS

*Joshua Senecal*
*Mark Duchaineau*
*Kenneth I. Joy*

This paper was submitted to the 7th IASTED International Conference on Computer Graphics and Imaging, to be held August 16–18, 2004, in Kauai, Hawaii

**February 20, 2004**

This research was supported under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

# REVERSIBLE $N$–BIT TO $N$–BIT INTEGER HAAR–LIKE TRANSFORMS

Joshua Senecal[*‡] Mark Duchaineau[†] Kenneth I. Joy[‡]
[*]Institute for Scientific Computing Research [‡]Center for Image Processing and Integrated Computing
[†]Center for Applied Scientific Computing Computer Science Department
Lawrence Livermore National Laboratory University of California, Davis

**ABSTRACT**
We introduce TLHaar, an $n$–bit to $n$–bit reversible transform similar to the Haar Integer Wavelet Transform (IWT). TLHaar uses lookup tables that approximate the Haar IWT, but reorder the coefficients so they fit into $n$ bits. TLHaar is suited for lossless compression in fixed–width channels, such as digital video channels and graphics hardware frame buffers.

**KEY WORDS**
Image Processing, Fixed–Width Transforms, Wavelets, Image Compression

## 1 Introduction

The Haar transform is probably the simplest and best known of the wavelet transforms [9]. It consists of a series of averaging and difference steps, each step operating on two adjacent values $A$ and $B$ and producing a low–pass value $\tilde{L} = (A + B)/2$ and a high–pass value $\tilde{H} = (A - B)/2$. The next iteration of the transform is performed on the low–pass values resulting from the previous iteration, and the process repeats until there is only a single low–pass value remaining. The original Haar transform produces floating–point coefficients due to the averaging step. For simplicity and speed, and to ensure that data is not lost due to the fixed–precision nature of floating–point hardware, a Haar integer wavelet transform (Haar IWT) approach is often used [7], where $\hat{L} = \lfloor (A + B)/2 \rfloor$ and $\hat{H} = B - A$[§].

A shortcoming of the Haar IWT is that, due to the subtraction that occurs in the transform procedure, it is necessary to store a sign bit for all nonzero high–pass coefficients. These sign bits present some problems. First, there is raw data inflation: the number of bits required to store the transformed data is greater than that required to store the original data. Second, since modern computers store data in 8-bit chunks, an actual implementation that takes (for example) 8–bit inputs must store each 9–bit coefficient in a data type 16 bits wide, thus requiring wider memory bandwidth. If this transform is being done in a fixed–width hardware environment and 16–bit values are unavailable some data loss will result.

In [10] a wavelet transform for binary images (bilevel, 1 bit per pixel) that overcomes these problems is described. We present one solution to these problems in the more general greylevel case: the Table–Lookup Haar (TLHaar) method, a reversible $n$–bit to $n$–bit transform. Because it is a fixed–width transform TLHaar is particularly suited for lossless compression and use in environments with fixed–width channels, such as digital video and graphics hardware frame buffers. TLHaar uses two lookup tables (LUTs) called AB2HL and HL2AB, each of size $2^{2n}$. These tables make the transform's range equal to that of the domain while preserving the most essential characteristics of the Haar transform. The closer that two inputs $A$ and $B$ are to each other in value, the closer their high–pass value is to zero, and the closer their low–pass value is to their true average.

To evaluate TLHaar we assembled a suite of 8–bit image sets, with each set containing images of a particular type (bilevel, shaded line art, photographs, etc.). We then implemented and optimized TLHaar and the Haar IWT. We recorded their execution times and compressed the coefficients they produced with several coders. Our tests indicate that TLHaar is up to 44% faster than Haar IWT, particularly when transforming data in large chunks. For data that have sharp edges, such as bilevel, line art, and computer generated images, coefficients generated by TLHaar compress better than Haar IWT. For other types of data, such as MRI, coefficients compress from 0.013% to 1.78% worse, depending on the compression method used.

## 2 Table–Lookup Haar

Table–Lookup Haar (TLHaar) performs a Haar–like transform, replacing the averaging and differencing steps with a single table lookup. TLHaar uses a set of two 2D lookup tables, called AB2HL and HL2AB. Each table contains $2^{2n}$ entries, with each entry being $2n$ bits wide, the upper and

[*]L–419, PO Box 808, Livermore, CA 94551, Tel: 925–422–3764, Fax: 925–422–7819, *senecal1@llnl.gov*
[†]L–561, PO Box 808, Livermore, CA 94551, Tel: 925–423–1320, FAX: 925–423–8704, *duchaine@llnl.gov*
[‡]One Shields Ave, Davis, CA 95616, Tel: 530–752–1077, FAX: 530–752–4767, *kijoy@ucdavis.edu*

[§]Throughout this paper $\tilde{H}$ and $\tilde{L}$ denote coefficients produced by Haar, $\hat{H}$ and $\hat{L}$ denote those produced by Haar IWT, and $H$ and $L$ denote those produced by our method, TLHaar.

lower $n$ bits each containing a value. AB2HL is used when performing a forward transform. It takes two $n$–bit data values $A$ and $B$ as indices and produces a $2n$ bit value, where the upper $n$ bits are the high–pass value $H$ and the lower $n$ bits are the low–pass value $L$. HL2AB is used when reversing a transform, similarly converting $(H, L)$ to $(A, B)$. The transform process is illustrated in figure 1.

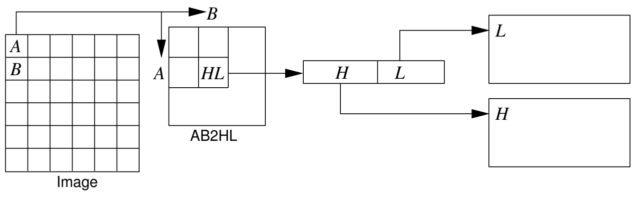


Figure 1. TLHaar transform process.

## 2.1 Transform LUTs

As shown in figure 2, the unscaled Haar transform expands the domain by $1/sin(45)$ and rotates it by 45 degrees. This gives the high– and low–pass values a range almost twice that of the original domain, as measured along the axes. Not all of the positions in the transform range are populated. The Haar IWT takes advantage of this, "squashing" the unscaled Haar range so that its low–pass values fall into the original domain. An $n$–bit to $n$–bit integer transform must additionally further squeeze the range so that its high–pass values also fall into the original domain.

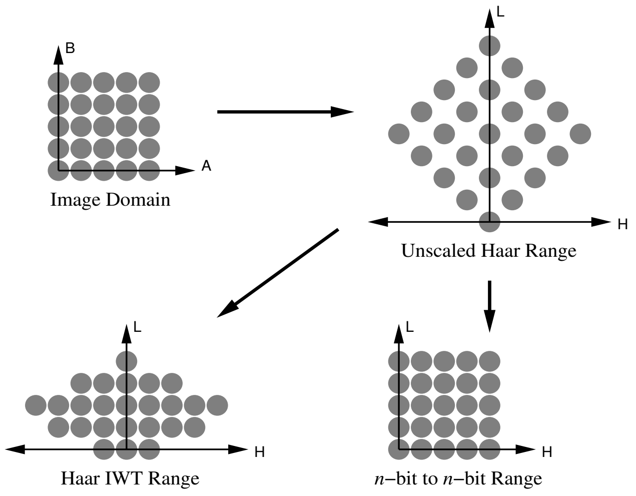


Figure 2. Domain and range of various transforms.

The LUTs used in TLHaar have the following properties:

$$|\tilde{H_i}| \leq |\tilde{H_j}| \iff |H_i| \leq |H_j| \quad (1)$$

$$|\tilde{L_i}| \leq |\tilde{L_j}| \iff |L_i| \leq |L_j|. \quad (2)$$

That is, for any given two pairs of data values their high– and low–pass values as created by TLHaar will have the same magnitude relationships as those created by Haar. For the transform table to be reversible there must be a 1:1 mapping between entries in the two tables. We therefore initialize each with an identity transform: AB2HL$[i, j]$ = $(i, j)$, HL2AB$[i, j]$ = $(i, j)$. We then rearrange the entries in HL2AB and AB2HL so properties 1 and 2 hold. We accomplish this via a sort of the LUTs according to the following pseudocode:

```
do {
    For each L Column in HL2AB
        Sort based on $|(A - B)/2.0|$
    For each H Row in HL2AB
        Sort based on $(A + B)/2.0$
} while (there was a swap)
```

During the sort process whenever a swap occurs in HL2AB the corresponding entries in AB2HL are also swapped.

It was not clear beforehand that this sort would converge. At this time we do not have a general proof that the sort will always converge, however we tested this process of creating tables for values $2 \leq n \leq 12$, and our tests indicate that in all cases the sort converges. We are unable to test further since when $n > 12$ the tables become so large they are impractical. For $n = 13$ a single LUT will contain over 67 million entries and be 256 megabytes in size.

Figure 3 shows the $(A, B) \rightarrow (H, L)$ and $(H, L) \rightarrow (A, B)$ transforms from the AB2HL and HL2AB LUTs after sorting, when $n = 8$. From these we see that TLHaar performs the desired transform: for example, $A$ and $B$ that are close in value have an $H$ close to zero as shown by the dark band on the diagonal of $(A, B) \rightarrow H$. Figure 4 gives the histogram of the "Lena" image before and after a TLHaar transform.

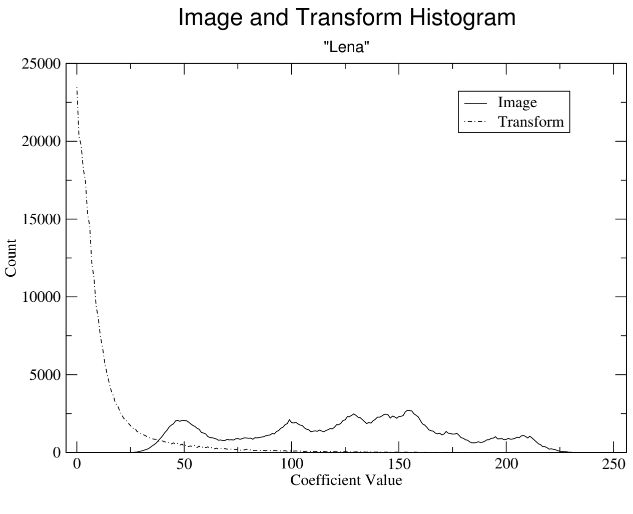


Figure 4. The histogram of the original pixels and the transform coefficients of the "Lena" image.

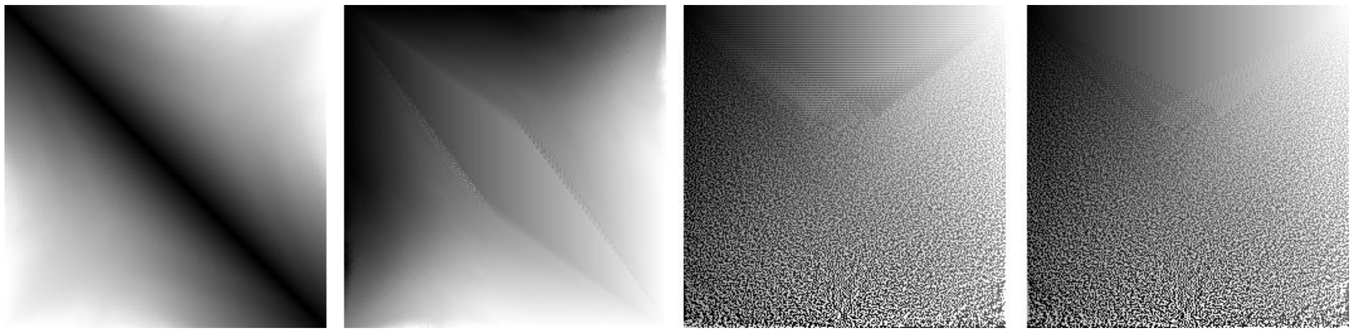


Figure 3. From left to right, the $(A, B) \rightarrow H$, $(A, B) \rightarrow L$, $(H, L) \rightarrow A$, and $(H, L) \rightarrow B$ transforms, as taken from the AB2HL and HL2AB tables. Black = 0, White = 255, (0,0) is the upper–left corner of each table.

## 2.2 Sorted LUTs Are Not Unique

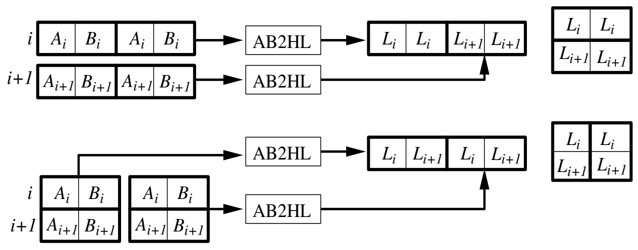
In section 2.1 we demonstrated the method used for creating the TLHaar LUTs, using a sort on an identity transform. If the LUTs are initialized with a 1:1 mapping different from our original method, will the sort still result in the same transform LUTs? The answer to this question is no, implying that LUTs satisfying relationships 1 and 2 are not unique. As a test we permuted one of the initialized LUTs before sorting, by randomly swapping entries. The corresponding entries in the other LUT were also swapped, maintaining a 1:1 mapping. We then performed our sort on the permuted tables. The resulting transform tables were different from those of our original method. By varying the number of swaps and the seed to the pseudorandom number generator we produced tables that were similar to each other, but not identical.

## 3 TLHaar Implementation Optimizations

Because TLHaar operates on and produces $n$–bit data, when $n$ is both a power of 2 and an integer size common in modern computer architectures (8–bit byte, 16–bit short integer, etc.) it is possible to store the low–pass and high–pass values in arrays of that integer type. This allows us to implement and take better advantage of some special optimizations. Here we describe optimizations made for an implementation that operates on 8–bit images.

We first altered how we perform table lookups in a row transform. Since input values $A$ and $B$ are adjacent in memory, instead of reading $A$ and $B$ separately and indexing the AB2HL LUT with both (i.e. $HL = AB2HL[A][B]$) we cast the input array of bytes into an array of 16–bit short integers, and read $A$ and $B$ together as a single short $AB$. This allows us to perform a complete table lookup using fewer operations: $HL = AB2HL[AB]$.

We would like to use the above optimization when performing a transform in the column direction. The standard row transform operates on an image one row at a time, writing out the resulting low–pass values such that they are contiguous in the row direction. Thus a given image column is not contiguous in memory. To solve this when performing a row transform we transform two rows at a time. Given the $k$-th pair of pixels from rows $i$ and $i+1$ we transform $A_iB_i$ and $A_{i+1}B_{i+1}$, and place $L_i$ and $L_{i+1}$ adjacent to each other in preparation for the column transform. The column transform can then proceed down columns in image space, but along adjacent memory locations. The idea behind the optimization is shown in figure 5. The right side shows the low–pass values ordered in both memory and image space, the bold lines indicating adjacent memory values.



i
$A_i$ $B_i$ $A_i$ $B_i$
i+1
$A_{i+1}$ $B_{i+1}$ $A_{i+1}$ $B_{i+1}$
AB2HL
AB2HL
$L_i$ $L_i$ $L_{i+1}$ $L_{i+1}$
$L_i$ $L_i$
$L_{i+1}$ $L_{i+1}$
AB2HL
AB2HL
$L_i$ $L_{i+1}$ $L_i$ $L_{i+1}$

Figure 5. The TLHaar row transform data reordering optimization. The top shows the normal transform procedure, and the bottom the reordering procedure.

## 4 Results

To obtain the results in this paper we assembled a suite of 8–bit images. The images fall into the following categories:

- BW_Lines. A collection of 106 bilevel (black–and–white) line figures. Each figure is small, an example size being 108 by 110 pixels [1].
- LineArt. A set of 18 line art images. Each image has shading, gradient fills, and the like.
- ObjectBank. A set of 122 computer–rendered images of everyday objects [1].
- MRI. A set of 185 MRI scans. Each image is 256 by 256 pixels [2].

- ccitt. Eight of the standard bilevel (black–and–white) ccitt FAX test images [3]. Each is 1728 by 2376 pixels.

- DB1_B. A set of 80 fingerprint scans[4]. Each is 300 by 300 pixels.

- DB2_B. A set of 80 fingerprint scans. These are the same fingerprints as in DB1_B, but each image is 256 by 364 pixels, and the images have been processed to bring out details.

- Photos. A set of 29 photographic images [5, 6]. These include standard test images (such as "Lena") and personal photographs from the lead author.

- r2_slices. A set of 221 randomly selected images extracted from the data produced by a Richtmeyer–Meshkov mixing simulation, described in [8].

- Power2. This is a selection of images from Photos, where each image is square and has an edge length that is a power of 2. These are used to evaluate the reordering method described in section 3.

In each table of results the column heading "% Gain" indicates gains obtained using TLHaar over the Haar IWT (positive percentage meaning TLHaar is better).

## 4.1 Execution Time

To compare execution times we implemented classes to perform the Haar IWT and TLHaar transforms on 8–bit grayscale images, and optimized each separately. We also implemented versions of TLHaar and the Haar IWT that perform data reordering and operate on square images with edge lengths that are a power of 2. Timings were taken on a 550 MHz PowerBook G4 running MacOS 10.1.5. The time given is an average over 10 transform runs, where a run transforms all images in a particular category, and includes only the time taken to transform the image. Execution times are given in table 1.

|  | Transform Time (sec) |  |  |
|---|---|---|---|
| Category | TLHaar | Haar IWT | % Gain |
| BW_Lines | 0.06368 | 0.06540 | 2.63 |
| LineArt | 0.15289 | 0.27621 | 44.65 |
| ObjectBank | 0.90947 | 1.08220 | 15.96 |
| MRI | 0.45442 | 0.55188 | 17.66 |
| MRI (reord) | 0.38121 | 0.51650 | 26.19 |
| ccitt | 1.31669 | 1.96779 | 33.09 |
| DB1_B | 0.25860 | 0.36670 | 29.48 |
| DB2_B | 0.31501 | 0.35577 | 11.46 |
| Photos | 1.02017 | 1.34319 | 24.05 |
| r2_slices | 0.31479 | 0.39773 | 20.85 |
| Power2 | 0.23322 | 0.31506 | 25.98 |
| Power2 (reord) | 0.20557 | 0.29665 | 30.70 |

Table 1. Our test image categories and their transform times. (reord) indicates execution time using the reordering method of section 3.

## 4.2 Compressibility of Coefficients

As our main interest in developing TLHaar was to create a fixed–width transform we are not greatly concerned about how efficiently TLHaar's coefficients compress compared to the Haar IWT. If the resulting compression ratio of TLHaar coefficients is within a percent or two of the rate obtained by the Haar IWT, we are content. To get a feel for how coefficients generated by TLHaar compress compared to those produced by the Haar IWT we transformed the test images and then compressed the results using three freely available compression programs: gzip[1], bzip[2], and an arithmetic coder available from Alistair Moffat[3]. We used binary and byte arithmetic encoding.

To gauge the effect of sign bits on the compressibility of Haar IWT coefficients we compressed them in two ways. In the first method the coefficient magnitudes were written as a stream of bytes and compressed, and the sign bit for each nonzero magnitude was appended uncompressed. In the second method coefficient magnitudes were written as a stream of bytes, and then a binary stream consisting of the sign bits of all nonzero magnitudes was appended. The combined stream was then compressed.

Due to lack of space we present in table 2 results only for the former method, as it presents Haar IWT more favorably and the comparison to TLHaar is more fair. Generally when the latter method is used Haar IWT's coefficients compress worse, to the extent that when byte arithmetic encoding is used TLHaar is always superior. Some example results in this case are TLHaar being 4.45% better in the Photos category and 7.23% better in the MRI category.

Results for TLHaar in table 2 are when using unpermuted tables, as described in section 2.1. We do not include the size of the transform LUTs in the TLHaar coefficient sizes. The tables are a static part of the transform process and are therefore known ahead of time, so in a coding application the tables do not need to be sent as part of the encoded data.

## 5 Conclusions and Future Research

The suitableness of TLHaar depends primarily on the need for protecting data against loss. If data needs to be processed losslessly in a fixed–width environment, TLHaar—by virtue of its fixed–width nature alone—is superior to the Haar IWT.

From our timing tests it appears that TLHaar is faster than the Haar IWT, particularly when data reordering is im-

[1]`http://www.gzip.org`

[2]`http://sources.redhat.com/bzip2/`

[3]`http://www.cs.mu.oz.au/~alistair/arith_coder/`

| Image | TLHaar | Haar IWT | % Gain | TLHaar | Haar IWT | % Gain |
|---|---|---|---|---|---|---|
| | gzip | | | bzip | | |
| BW_Lines | 360972 | 466731 | 22.66 | 391737 | 500173 | 21.68 |
| LineArt | 419235 | 528964 | 20.74 | 445387 | 515182 | 13.55 |
| ObjectBank | 3131056 | 3341803 | 6.31 | 3115625 | 3258474 | 4.38 |
| MRI | 5477859 | 5381934 | -1.78 | 5281869 | 5190000 | -1.77 |
| ccitt | 1157989 | 1557889 | 25.67 | 1013614 | 1454085 | 30.29 |
| DB1_B | 5016509 | 5086675 | 1.38 | 4973728 | 4853424 | -2.48 |
| DB2_B | 5820744 | 6165921 | 5.60 | 6175512 | 6353409 | 2.80 |
| Photos | 15029496 | 14912472 | -0.78 | 14963474 | 14651157 | -2.13 |
| r2_slices | 777975 | 801756 | 2.97 | 785564 | 783836 | -0.22 |
| | Binary Arithmetic | | | Byte Arithmetic | | |
| BW_Lines | 391472 | 494289 | 20.80 | 377422 | 496202 | 23.94 |
| LineArt | 509478 | 627262 | 18.78 | 535573 | 677915 | 21.00 |
| ObjectBank | 3290576 | 3474223 | 5.29 | 3620160 | 3771556 | 4.01 |
| MRI | 5691930 | 5627898 | -1.14 | 5408487 | 5407783 | -0.013 |
| ccitt | 1199500 | 1652339 | 27.41 | 1288297 | 1969556 | 34.59 |
| DB1_B | 5673653 | 5346789 | -6.11 | 4796648 | 4703212 | -1.99 |
| DB2_B | 6415074 | 6803730 | 5.71 | 5679509 | 5946472 | 4.49 |
| Photos | 15234370 | 14873308 | -2.43 | 14285797 | 14090810 | -1.38 |
| r2_slices | 828160 | 850696 | 2.65 | 823509 | 834297 | 1.29 |

Table 2. Compressed category sizes (in bytes).

plemented. The only case in which TLHaar's speed is close to that of Haar IWT is when the images are rather small, as is the case with the BW_Lines image set. Although the best case 44% speed increase in our tests may not appear to be a large gain, we point out that this speed gain is being taken against a wavelet transform that is already very fast. These speed results are promising, but further tests on different hardware platforms are needed to obtain a better indication of TLHaar's execution time.

From our simple compression tests TLHaar appears to be superior when the data being processed is bilevel, or contains lines or hard edges (as are in the BW_Lines, LineArt, ObjectBank, ccitt, and DB2_B sets). For other classes of images the results are mixed, and vary depending on the coding method used. For example, when using gzip as the compressor, the TLHaar coefficients produced for the Photos category compress 0.78% worse than the Haar IWT coefficients. The gap widens to 2.43% worse when binary arithmetic coding is used. Conversely for the MRI data set gzip compresses TLHaar coefficients 1.78% worse than Haar IWT coefficients, but using byte arithmetic encoding this gap narrows to only 0.013% worse. Given that TLHaar was designed to operate in a fixed–width environment and the Haar IWT was not, we feel these compression rates are acceptable.

We do not expect our current implementation of TLHaar to be very useful for lossy compression. Future research will focus on developing variations of TLHaar that are more suitable for lossy techniques.

Other research will center around studying the LUTs in more detail. During the transform process each image type only touches a small percentage of the total entries in the LUT. These entries are often in clusters. It may therefore be possible to create LUTs for specific data types by optimizing only those parts of the LUT that the image touches.

## 6 Acknowledgements

Images in our test suite were provided by a variety of sources. In particular, the BW_Lines and ObjectBank images were obtained from Michael J. Tarr at Brown University [1]. The LineArt images are clip art downloaded through AppleWorks, from Apple Computer, Inc. Some images in the Photos category were obtained from the Signal and Image Processing Institute at the University of Southern California [5], and others from the Waterloo BragZone maintained by John Kominek [6]. MRI data was provided by the Department of Radiology at UCSD Medical Center, and the Vision List Imagery archive [2]. Fingerprint images were obtained from the web site of the Fingerprint Verification Competition 2000 [4]. We are grateful to all those who made data available.

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Joshua Senecal's work was supported in part by a United States Department of Education Government Assistance in Areas of National Need (DOE-GAANN) grant #P200A980307.

## References

[1] `http://www.cog.brown.edu/~tarr/`

stimuli.html.

[2] ftp://ftp.vislist.com/IMAGERY/MED_3D_SLICES.

[3] ftp://ftp.funet.fi/pub/graphics/misc/test-images/.

[4] http://bias.csr.unibo.it/fvc2000/download.asp.

[5] http://sipi.usc.edu/services/database/.

[6] http://links.uwaterloo.ca/bragzone.base.html.

[7] Calderbank, Daubechies, Sweldens, and Yeo. Lossless image compression using integer to integer wavelet transforms. In *International Conference on Image Processing*, pages 596–599. IEEE Computer Society, 1997.

[8] Mirin, Cohen, Curtis, Dannevik, Dimits, Duchaineau, Eliason, Schikore, Anserson, Porter, Woodward, Shieh, and White. Very high resolution simulation of compressible turbulence on the ibm–sp system. Technical Report UCRL–JC–134237, Lawrence Livermore National Laboratory, 1999.

[9] Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann, second edition, 2000.

[10] Mitchell Swanson and Ahmed Tewfik. A binary wavelet decomposition of binary images. *IEEE Transactions on Information Processing*, 3(12):1637–1650, Dec 1996.